GAMECENTRIC

È l'ora di <ranges>

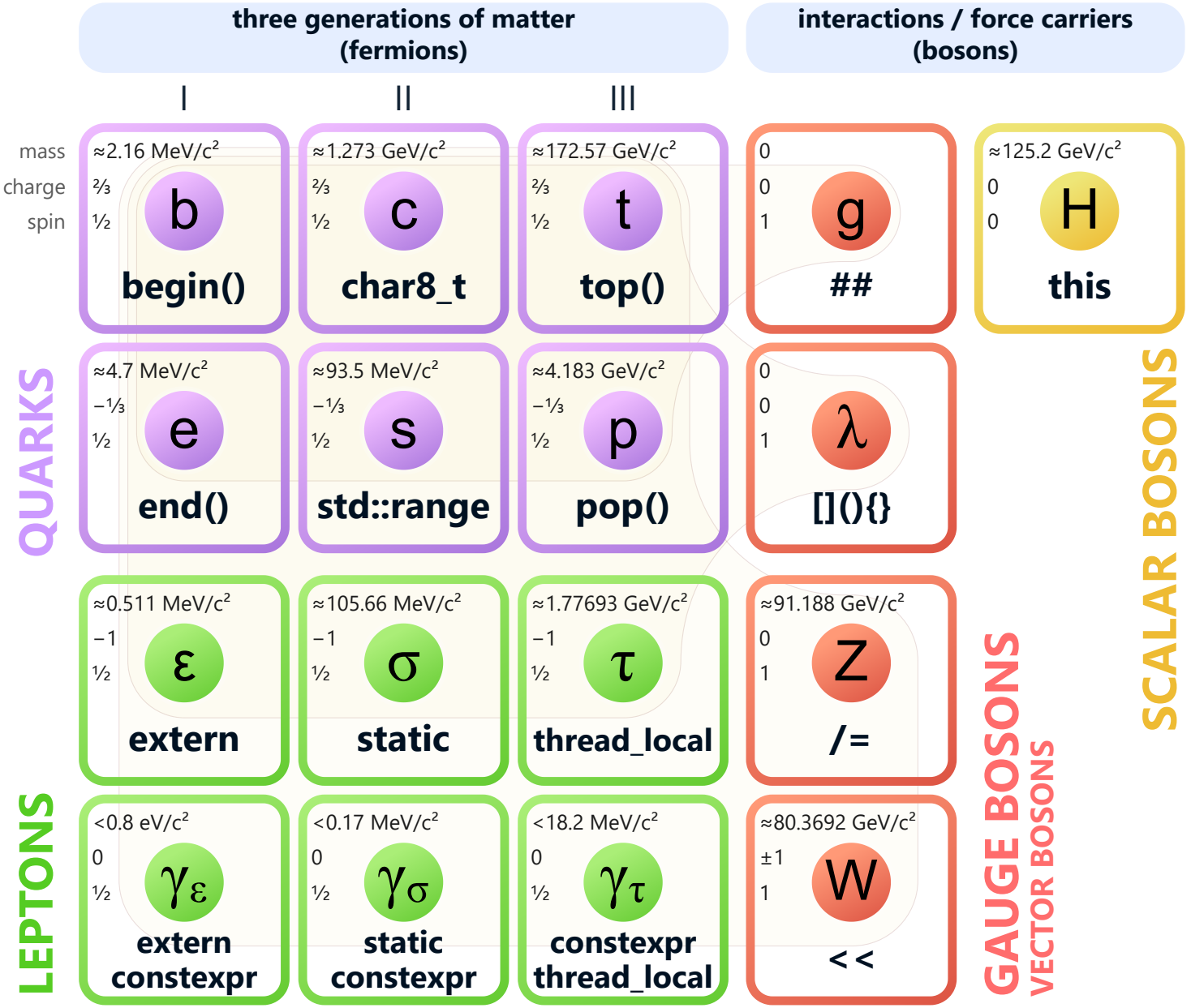
Alberto Barbati (Nacon Studio Milan)

Meetup itC++ 16/04/2025

Alberto Barbati

- Programmatore C++ entusiasta dal 1990
- Video game developer dal 2001, ha collaborato con Ubisoft, Digital Tales, Reply Game Studios e Nacon Studio Milan
- Trainer di Game Programming presso Digital Bros Game Academy
- Segue con i lavori della C++ Committee dal 2008

Standard Model of C++20



Programma di oggi

- Le principali novità in `<ranges>` e in `<algorithm>` introdotte da C++20
- Come usare la libreria con contenitori non-standard: l'esempio di Unreal Engine

'Begin at the beginning,'
the King said gravely,
'and go on till you come to the end:
then stop.'

– Alice's Adventures in Wonderland

Intervallo

```
// Intervallo [inizio, fine)

for (auto it = inizio; it != fine; ++it)
{
    // uso *it
}
```

- In C++17 un intervallo è definito come coppia di iteratori
- In C++20 la definizione viene rilassata: un intervallo è una coppia iteratore/sentinella, dove l'unico requisito sulla sentinella è che sia confrontabile con l'iteratore

<algorithm> in C++17

```
namespace std
{
    template <class RandomAccessIterator>
    void sort(RandomAccessIterator first, RandomAccessIterator last);
}
```

23.3.5 C++17 iterator requirements [iterator.cpp17]

23.3.5.1 General [iterator.cpp17.general]

1 In the following sections, `a` and `b` denote values of type `X` or `const X`, `difference_type` and `reference` refer to the types `iterator_traits<X>::difference_type` and `iterator_traits<X>::reference`, respectively, `n` denotes a value of `difference_type`, `u`, `tmp`, and `m` denote identifiers, `r` denotes a value of `X&`, `t` denotes a value of value type `T`, `o` denotes a value of some type that is writable to the output iterator.

[*Note 1*: For an iterator type `X` there must be an instantiation of `iterator_traits<X>` (23.3.2.3). — *end note*]

23.3.5.2 *Cpp17Iterator* [iterator.iterators]

1 The *Cpp17Iterator* requirements form the basis of the iterator taxonomy; every iterator meets the *Cpp17Iterator* requirements. This set of requirements specifies operations for dereferencing and incrementing an iterator. Most algorithms will require additional operations to read (23.3.5.3) or write (23.3.5.4) values, or to provide a richer set of iterator movements (23.3.5.5, 23.3.5.6, 23.3.5.7).

2 A type `X` meets the *Cpp17Iterator* requirements if:

(2.1) — `X` meets the *Cpp17CopyConstructible*, *Cpp17CopyAssignable*, and *Cpp17Destructible* requirements (16.4.4.2) and lvalues of type `X` are swappable (16.4.4.3), and

(2.2) — `iterator_traits<X>::difference_type` is a signed integer type or `void`, and

(2.3) — the expressions in Table 84 are valid and have the indicated semantics.

Table 84: *Cpp17Iterator* requirements [tab:iterator]

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
`*r`	unspecified		*Preconditions*: `r` is dereferenceable.
`++r`	`X&`		

Se i requisiti non sono soddisfatti?

```
float begin, end;
std::sort(begin, end);
```

Ben 106 righe di messaggi di errore.

```
In file included from <source>:3:
In file included from /include/c++/12.2.0/algorithm:61:
/include/c++/12.2.0/bits/stl_algo.h:1938:5: error: call to '__lg' is ambiguous
                                std::__lg(__last - __first) * 2,
                                ^~~~~~~~
/include/c++/12.2.0/bits/stl_algo.h:4820:12: note: in instantiation of function template specialization 'std::__sort<float
      std::__sort(__first, __last, __gnu_cxx::__ops::__iter_less_iter());
           ^
<source>:8:10: note: in instantiation of function template specialization 'std::sort<float>' requested here
    std::sort(begin, end);
         ^
/include/c++/12.2.0/bits/stl_algobase.h:1505:3: note: candidate function
  __lg(int __n)
  ^
/include/c++/12.2.0/bits/stl_algobase.h:1509:3: note: candidate function
  __lg(unsigned __n)
  ^
/include/c++/12.2.0/bits/stl_algobase.h:1513:3: note: candidate function
  __lg(long __n)
  ^
/include/c++/12.2.0/bits/stl_algobase.h:1517:3: note: candidate function
  __lg(unsigned long __n)
  ^
/include/c++/12.2.0/bits/stl_algobase.h:1521:3: note: candidate function
  __lg(long long __n)
  ^
/include/c++/12.2.0/bits/stl_algobase.h:1525:3: note: candidate function
  __lg(unsigned long long __n)
  ^
In file included from <source>:2:
In file included from /include/c++/12.2.0/memory:63:
In file included from /include/c++/12.2.0/bits/stl_algobase.h:71:
/include/c++/12.2.0/bits/predefined_ops.h:45:16: error: indirection requires pointer operand ('float' invalid)
      { return *__it1 < *__it2; }
               ^~~~~~
/include/c++/12.2.0/bits/stl_algo.h:1809:8: note: in instantiation of function template specialization '__gnu_cxx::__ops::
        if (__comp(__i, __first))
```

C++20 introduce i concept

I requisiti si possono ora esprimere direttamente nel codice

```
template <class I>
concept input_or_output_iterator =
    weakly_incrementable<I> &&
    requires(I i)
    {
        { *i } -> can-reference;
    };
```

<algorithm> in C++20

A `std::sort` si affianca `std::ranges::sort`

```
namespace std::ranges
{
    template <random_access_iterator I, sentinel_for<I> S, /* ... */>
        requires sortable<I, /* ... */>
    /* ... */ sort(I first, S last, /* ... */);
}
```

Requisiti non soddisfatti in C++20

```
float begin, end;
std::ranges::sort(begin, end);
```

```
<source>:8:5: error: no matching function for call to object of type 'const __sort_fn'
    std::ranges::sort(begin, end);
    ^~~~~~~~~~~~~~~~~
/include/c++/12.2.0/bits/ranges_algo.h:1814:7: note: candidate template ignored: constraints not satisfied [with _Ite
      operator()(_Iter __first, _Sent __last,
      ^
/include/c++/12.2.0/bits/ranges_algo.h:1810:14: note: because 'float' does not satisfy 'random_access_iterator'
    template<random_access_iterator _Iter, sentinel_for<_Iter> _Sent,
             ^
/include/c++/12.2.0/bits/iterator_concepts.h:662:38: note: because 'float' does not satisfy 'bidirectional_iterator'
    concept random_access_iterator = bidirectional_iterator<_Iter>
```

Proiezioni

La maggior parte degli algoritmi hanno un parametro aggiuntivo, detto *proiezione*

```
struct person
{
    std::string name;
    int age;
};

void example(std::vector<person>& dati)
{
    std::ranges::sort(dati, {}, &person::age);

    auto it = std::ranges::find(dati, "Alberto", &person::name);
}
```

Intervalli come unico argomento

Tutti i nuovi algoritmi di C++20 sono presenti sia nella variante "classica" con la coppia di iteratori, sia nella variante "moderna" con un solo argomento per l'intero intervallo

```
void example(std::vector<int>& dati)
{
    // C++17
    std::sort(dati.begin(), dati.end());

    // C++20
    std::ranges::sort(dati.begin(), dati.end());

    // oppure
    std::ranges::sort(dati);
}
```

Da grandi poteri derivano grandi responsabilità

```
void example()
{
    bool all_tests = std::ranges::all_of(tests()); // 😍

    std::ranges::for_each(get_data(), process_data); // 👍

    std::ranges::sort(get_data()); // 🤨

    auto it = std::ranges::find(get_data(), data_to_find); // 😢
    if (it != get_data().end())
    {
        // *it
    }
}
```

Riduzione al problema precedente

La variante moderna è implementata in termini della variante classica:

```
std::ranges::sort(data)
```

è di fatto equivalente a

```
std::ranges::sort(std::ranges::begin(data), std::ranges::end(data))
```

e similmente per tutti gli altri algoritmi

Customization point object

`std::ranges::begin` e `std::ranges::end` sono le porte di ingresso della libreria ranges tanto che il concept `range` è definito così:

```
namespace std::ranges
{
    template<class T>
    concept range = requires(T& t)
    {
        ranges::begin(t);
        ranges::end(t);
    };
}
```

`std::ranges::begin(rng)`

1. se `rng` è un rvalue → [...]
2. se `rng` è un array → l'indirizzo al primo elemento dell'array
3. se `rng.begin()` esiste e soddisfa `input_or_output_iterator` → `rng.begin()`
4. se `begin(rng)` esiste e soddisfa `input_or_output_iterator` → `begin(rng)`
5. Altrimenti è ill-formed

`std::ranges::end(rng)`

1. `rng` è un rvalue → [...]
2. `rng` è un array → l'indirizzo past-the-end dell'array
3. `rng.end()` esiste e soddisfa `sentinel_for<B>` → `rng.end()`
4. `end(rng)` esiste e soddisfa `sentinel_for<B>` → `end(rng)`
5. Altrimenti è ill-formed

dove `B` è `decltype(std::ranges::begin(rng))`

Considerazioni

- `rng` è un `range` se e solo se `std::ranges::begin(rng)` e `std::ranges::end(rng)` sono definiti e, rispettivamente, un iteratore e una sentinella adatta all'iteratore
- Tutti i contenitori della libreria C++ sono `range`
- Contenitori di altre librerie possono diventare `range` implementando correttamente le giuste funzioni

Contenitori non standard: Unreal

Unreal Engine utilizza una sua libreria di contenitori alternativi a quelli della libreria standard C++, i più usati sono `TArray`, `TSet` e `TMap` che corrispondono, grosso modo, a `std::vector`, `std::unordered_set` e `std::unordered_map`

Purtroppo questi contenitori non si possono usare con la libreria `<ranges>` out-of-the-box, vediamo perché e come è possibile ovviare

Iterare i contenitori Unreal

Nativamente, per iterare i contenitori di Unreal si usa un pattern con un solo iteratore

```
TArray<int> Array;

for (auto It = Array.CreateIterator(); It; ++It)
{
    // int& Element = *It;
}
```

Iterazione range-based

Nelle versioni più recenti di Unreal è stato implementato anche il supporto per il range-based loop, aggiungendo le opportune funzioni membro `begin()` e `end()`, ma...

```
TArray<int> Array;

for (int& Element : Array)
{
    /* ... */
}
```

Sebbene `TArray` abbia metodi `begin()` e `end()`, questo non lo rende un `range`, perché...

Fare le cose a metà

```
// This iterator type only supports the minimal functionality needed to support
// C++ ranged-for syntax.  For example, it does not provide post-increment ++ nor ==.
```

... gli iteratori restituiti dai metodi `begin()` e `end()` dei contenitori Unreal non soddisfano i giusti concept e quindi `std::ranges::begin` e `std::ranges::end` sono ill-formed!

TArray

Nel caso specifico di `TArray` (ma anche `TString`), è possibile sfruttare il fatto che gli elementi sono conservati in memoria in modo contiguo

```
template <typename T, typename Allocator>
auto begin(TArray<T, Allocator>& Array)
{
    return Array.GetData();
}

template <typename T, typename Allocator>
auto end(TArray<T, Allocator>& Array)
{
    return Array.GetData() + Array.Num();
}
```

Caso generale

Invece per `TSet` e `TMap` l'idea è di usare l'iteratore nativo del contenitore Unreal in combinazione con una sentinella

```
template <typename T, typename K, typename A>
auto begin(TSet<T, K, A>& Set)
{
    return Set.CreateIterator();
}

template <typename T, typename K, typename A>
auto end(TSet<T, K, A>& Array)
{
    return std::default_sentinel_t{};
}
```

Problemi

Vorremmo quindi definire `operator==` così

```
template <typename T, typename K, typename A>
bool operator==(const TSet<T, K, A>::TIterator& It, std::default_sentinel_t)
{
    return !It;
}
```

ma non è possibile, perchè il primo argomento non consente di dedurre `T`, `K` e `A`

Inoltre per una scelta scellerata, `TIterator` ha un membro reference che lo rende non-copiabile e non-assegnabile, pertanto non soddisfa i concept necessari

Wrapper

```
template <typename T, typename K, typename A>
struct SetIterator
{
    typename TSet<T, K, A>::TIterator Base;

    SetIterator(typename TSet<T, K, A>::TIterator It);
    SetIterator(const SetIterator&);
    SetIterator& operator=(const SetIterator&);
    SetIterator& operator++();
    SetIterator operator++(int);
    T& operator*() const;
    friend bool operator==(const SetIterator&, std::default_sentinel_t);
};
```

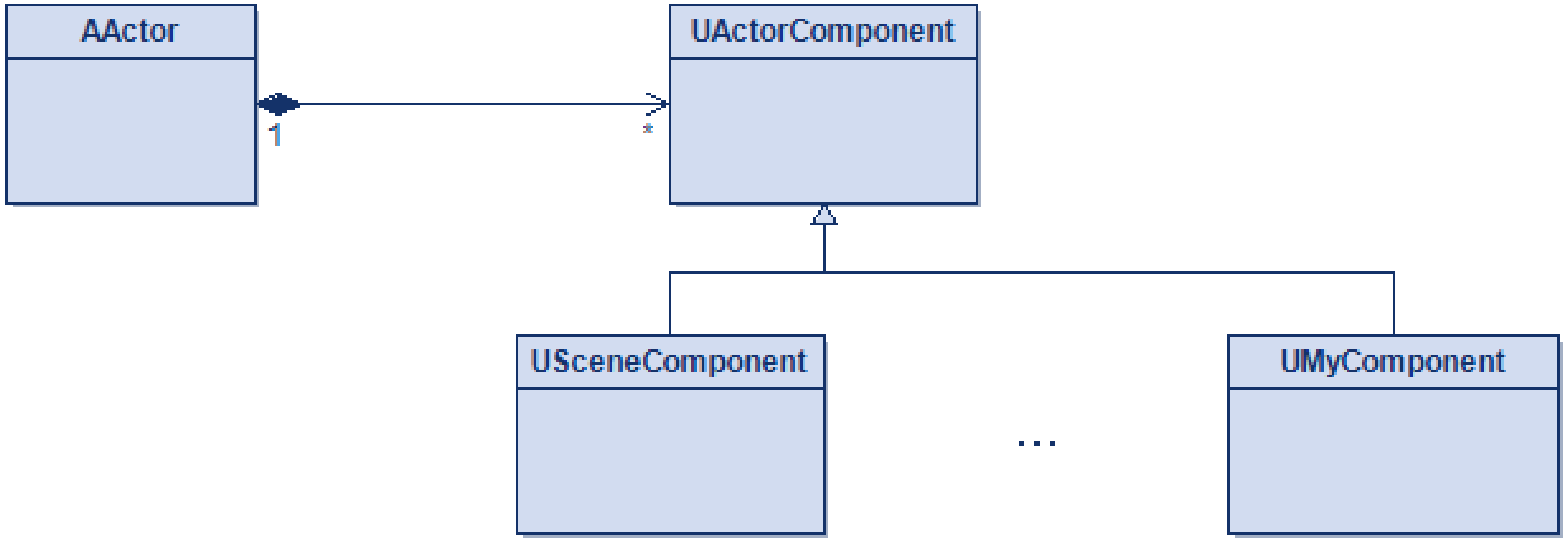
Range adaptor

Un *range adaptor* è un range che applica delle operazioni *lazy* ad un range esistente

Tramite i range adaptor, è possibile eseguire operazioni senza necessariamente memorizzare i risultati intermedi in contenitori

I range adaptor possono essere facilmente composti per realizzare operazioni complesse a partire da operazioni base

Unreal Actors and Components



Problema: iterare i componenti di un attore

```
for (auto Component : Actor->GetComponents())
{
    if (auto MyComponent = Cast<UMyComponent>(Component))
    {
        // MyComponent è UMyComponent*
    }
}
```

std::views::transform

`std::views::transform` applica una funzione ad ogni elemento del range, restituendo un range dei risultati

```
constexpr auto MyCast = [](auto* Comp) { return Cast<UMyComponent>(Comp); };

for (auto Component : Actor->GetComponents() | std::views::transform(MyCast))
{
    // Component è UMyComponent*
}
```

In <ranges> tutto è oggetto

```
constexpr auto MyCast = std::views::transform(
    [](auto* Comp) { return Cast<UMyComponent>(Comp); });

for (auto Component : Actor->GetComponents() | MyCast)
{
    // Component è UMyComponent*
}
```

std::views::filter

Con `std::views::filter` possiamo selezionare gli elementi del range che soddisfano ad un predicato, scartando gli altri

```
constexpr auto MyCast = std::views::transform(
    [](auto* Comp) { return Cast<UMyComponent>(Comp); });

constexpr auto NotNull = std::views::filter(
    [](auto* Ptr) { return Ptr != nullptr; });

for (auto Component : Actor->GetComponents() | MyCast | NotNull)
{
    // Component è UMyComponent*
}
```

Spostiamo tutto in una libreria

```
// RangesTools.h

namespace NSM
{
    template <typename Type>
    inline constexpr auto Cast = std::views::transform(
        [](auto* Ptr) { return ::Cast<Type>(Ptr); });

    inline constexpr auto NotNull = std::views::filter(
        [](auto* Ptr) { return Ptr != nullptr; });

    template <typename Type>
    inline constexpr auto CastValid = Cast<Type> | NotNull;
}
```

Risultato finale

```
for (auto Component : Actor->GetComponents() | NSM::CastValid<UMyComponent>)
{
    // Component è UMyComponent*
}
```

L'appetito vien mangiando

```
namespace NSM
{
    template <typename Type>
    inline constexpr auto GetComponents = [](AActor* Actor)
    {
        return Actor->GetComponents() | NSM::CastValid<UMyComponent>;
    };
}
```

```
for (auto Component : NSM::GetComponents<UMyComponent>(Actor))
{
    // Component è UMyComponent*
}
```

Un esempio più complesso

```
TArray<AActor*> Actors = /* ... */

for (auto Actor : Actors)
{
    for (auto Component : NSM::GetComponents<UMyComponent>(Actor))
    {
        // Component è UMyComponent*
    }
}
```

std::views::join

`std::views::join` prende un range di range di `T` e restituisce un range di `T`

```
TArray<AActor*> Actors = /* ... */

for (auto Component : Actors
        | std::views::transform(NSM::GetComponents<UMyComponent>)
        | std::views::join)
{
    // Component è UMyComponent*
}
```

Range adaptor: selezione

- `std::views::take` prende i primi `N` elementi da un range e scarta il resto
- `std::views::drop` scarta i primi `N` elementi da un range e prende il resto
- `std::views::take_while` e `std::views::drop_while` fanno lo stesso ma invece di contare gli elementi, procedono finché un predicato è vero
- `std::views::reverse` itera un range in senso inverso

`std::views::XXX` vs. `std::ranges::XXX_view`

La libreria propone sia `std::views::reverse`, sia `std::ranges::reverse_view` e similmente per molti range adaptor

Solitamente si usa `std::views::reverse` che "fa la cosa giusta" che solitamente utilizza `std::ranges::reverse_view` e a volte... no.

Range adaptor: accesso ai componenti

- `std::views::keys` itera un range di coppie chiave/valore, restituendo la sola chiave
- `std::views::values` itera un range di coppie chiave/valore, restituendo il solo valore
- `std::views::elements` itera un range di tuple, restituendo la componente n-esima

Altri range adaptor

- `std::views::split` e `std::views::lazy_split` suddividono un range in subrange, in base ad un delimitatore
- `std::views::as_rvalue` casta ogni elemento a rvalue
- `std::views::as_const` casta ogni elemento a const

Grazie dell'attenzione!

alberto@gamecentric.com

Credits

- Immagine della slide 2: Rielaborazione di https://commons.wikimedia.org/w/index.php?title=File:Standard_Model_of_Elementary_Particles.svg
- Immagine della slide 4: By John Tenniel - This file has been provided by the British Library from its digital collections. It is also made available on a British Library website. Catalogue entry: Cup.410.g.74., Public Domain, https://commons.wikimedia.org/w/index.php?curid=37072115